LoRA

Copy page

> This is experimental and the API may change in the future.

LoRA (Low-Rank Adaptation of Large Language Models) is a popular and lightweight training technique that significantly reduces the number of trainable parameters. It works by inserting a smaller number of new weights into the model and only these are trained. This makes training with LoRA much faster, memory-efficient, and produces smaller model weights (a few hundred MBs), which are easier to store and share. LoRA can also be combined with other training techniques like DreamBooth to speedup training.

> LoRA is very versatile and supported for DreamBooth, Kandinsky 2.2, Stable Diffusion XL, text-to-image, and Wuerstchen.

This guide will explore the train_text_to_image_lora.py script to help you become more familiar with it, and how you can adapt it for your own use-case.

Before running the script, make sure you install the library from source:

```
git clone https://github.com/huggingface/diffusers
cd diffusers
pip install .
```

Navigate to the example folder with the training script and install the required dependencies for the script you're using:

```
cd examples/text_to_image
pip install -r requirements.txt
```

Ask HuggingChat

> 🤗 Accelerate is a library for helping you train on multiple GPUs/TPUs or with mixed-precision. It'll automatically configure your training setup based on your hardware and environment. Take a look at the 🤗 Accelerate Quick tour to learn more.

Initialize an 🤗 Accelerate environment:

```
accelerate config
```

To setup a default 🤗 Accelerate environment without choosing any configurations:

```
accelerate config default
```

Or if your environment doesn't support an interactive shell, like a notebook, you can use:

```
from accelerate.utils import write_basic_config

write_basic_config()
```

Lastly, if you want to train a model on your own dataset, take a look at the Create a dataset for training guide to learn how to create a dataset that works with the training script.

> The following sections highlight parts of the training script that are important for understanding how to modify it, but it doesn't cover every aspect of the script in detail. If you're interested in learning more, feel free to read through the script and let us know if you have any questions or concerns.

Script parameters

The training script has many parameters to help you customize your training run. All of the parameters and their descriptions are found in the `parse_args()` function. Default values are provided for most parameters that work pretty well, but you can also set your own values in the training command if you'd like.

For example, to increase the number of epochs to train:

```
accelerate launch train_text_to_image_lora.py \
  --num_train_epochs=150 \
```

Many of the basic and important parameters are described in the Text-to-image training guide, so this guide just focuses on the LoRA relevant parameters:

- `--rank`: the inner dimension of the low-rank matrices to train; a higher rank means more trainable parameters
- `--learning_rate`: the default learning rate is 1e-4, but with LoRA, you can use a higher learning rate

Training script

The dataset preprocessing code and training loop are found in the `main()` function, and if you need to adapt the training script, this is where you'll make your changes.

As with the script parameters, a walkthrough of the training script is provided in the Text-to-image training guide. Instead, this guide takes a look at the LoRA relevant parts of the script.

UNet | text encoder

Diffusers uses `LoraConfig` from the PEFT library to set up the parameters of the LoRA adapter such as the rank, alpha, and which modules to insert the LoRA weights into. The adapter is added to the UNet, and only the LoRA layers are filtered for optimization in `lora_layers`.

```
unet_lora_config = LoraConfig(
    r=args.rank,
    lora_alpha=args.rank,
    init_lora_weights="gaussian",
    target_modules=["to_k", "to_q", "to_v", "to_out.0"],
)

unet.add_adapter(unet_lora_config)
lora_layers = filter(lambda p: p.requires_grad, unet.parameters())
```

The optimizer is initialized with the `lora_layers` because these are the only weights that'll be optimized:

```
optimizer = optimizer_cls(
    lora_layers,
    lr=args.learning_rate,
    betas=(args.adam_beta1, args.adam_beta2),
    weight_decay=args.adam_weight_decay,
```

```
    eps=args.adam_epsilon,
)
```

Aside from setting up the LoRA layers, the training script is more or less the same as train_text_to_image.py!

Launch the script

Once you've made all your changes or you're okay with the default configuration, you're ready to launch the training script! 🚀

Let's train on the Naruto BLIP captions dataset to generate your own Naruto characters. Set the environment variables `MODEL_NAME` and `DATASET_NAME` to the model and dataset respectively. You should also specify where to save the model in `OUTPUT_DIR`, and the name of the model to save to on the Hub with `HUB_MODEL_ID`. The script creates and saves the following files to your repository:

- saved model checkpoints
- `pytorch_lora_weights.safetensors` (the trained LoRA weights)

If you're training on more than one GPU, add the `--multi_gpu` parameter to the `accelerate launch` command.

> A full training run takes ~5 hours on a 2080 Ti GPU with 11GB of VRAM.

```
export MODEL_NAME="stable-diffusion-v1-5/stable-diffusion-v1-5"
export OUTPUT_DIR="/sddata/finetune/lora/naruto"
export HUB_MODEL_ID="naruto-lora"
export DATASET_NAME="lambdalabs/naruto-blip-captions"

accelerate launch --mixed_precision="fp16"  train_text_to_image_lora.py \
  --pretrained_model_name_or_path=$MODEL_NAME \
  --dataset_name=$DATASET_NAME \
  --dataloader_num_workers=8 \
  --resolution=512 \
  --center_crop \
  --random_flip \
  --train_batch_size=1 \
  --gradient_accumulation_steps=4 \
  --max_train_steps=15000 \
  --learning_rate=1e-04 \
  --max_grad_norm=1 \
```

```
  --lr_scheduler="cosine" \
  --lr_warmup_steps=0 \
  --output_dir=${OUTPUT_DIR} \
  --push_to_hub \
  --hub_model_id=${HUB_MODEL_ID} \
  --report_to=wandb \
  --checkpointing_steps=500 \
  --validation_prompt="A naruto with blue eyes." \
  --seed=1337
```

Once training has been completed, you can use your model for inference:

```
from diffusers import AutoPipelineForText2Image
import torch

pipeline = AutoPipelineForText2Image.from_pretrained("stable-diffusion-v1-5/stabl
pipeline.load_lora_weights("path/to/lora/model", weight_name="pytorch_lora_weight
image = pipeline("A naruto with blue eyes").images[0]
```

Next steps

Congratulations on training a new model with LoRA! To learn more about how to use your new model, the following guides may be helpful:

- Learn how to load different LoRA formats trained using community trainers like Kohya and TheLastBen.
- Learn how to use and combine multiple LoRA's with PEFT for inference.

</> Update on GitHub

← DreamBooth

Custom Diffusion →